

Improving the Superset Adder: A Fault-tolerant, Reconfigurable Parallel Prefix Adder

Kyle E. Powers | Dar-Eaum A. Nam | Eric A. Llana

ECE 4332 – Fall 2012

University of Virginia

<kep4md | dan6ra | eal2yf>@virginia.edu

ABSTRACT

In this paper, we visit the concept of a fault-tolerant, reconfigurable parallel prefix adder dubbed the Superset Adder. We prove its functionality through the use of tools such as Logisim and Cadence. We also offer, through verification in simulation, modifications in improving the Superset Adder's delay, power consumption, and area. Our results show an average 8.2% increase in performance and 8.1% decrease in power consumption, depending on configuration. Our extensions lay out various ways to further decrease delay, power, and area.

1. INTRODUCTION

Technology scaling increases process, voltage, and temperature, or PVT, variations, aging phenomena, and to worse hard and soft error rates [1]. This necessitates robust system and circuit design methodologies whilst using unreliable components. Key aspects of such design methodologies are redundancy, error detection and correction, and reconfiguration. These techniques have long been prevalent in hardware with two-dimensional regularity, such as memory, but are now essential in random logic and hardware with one-dimensional regularity, such as datapaths. Robustness for random logic requires massive redundancy, such as triple-modular redundancy, or TMR [2]. Robustness for circuits with one-dimensional regularity proves difficult, as two-dimensional schemes like parity and error correction code, or ECC, are not applicable.

The key to connecting robustness to datapath circuits, such as adders, is through exploiting their inherent redundancy. Yes, adders have been widely explored in the past and the parallel prefix adder (PPA) bears no exception [3,4]; however, new design spaces and PPA topologies have been recently introduced that are better geared toward robust applications. PPAs are important because a prefix, or dot, operation is constructed that permits the computation of intermediate carries. In conjunction with *generate* and *propagate* signals, the prefix operator allows PPAs to obtain an advantageous latency of $O(\log_2 N)$ instead of $O(N)$, like in a Ripple Carry adder.

Guevara and Gregg [5], for example, utilize the inherent redundancy in a Kogge-Stone to develop a fault-tolerant, real-time reconfigurable PPA. In [6], Bailey and Stan create a new taxonomy for fault-tolerant, reconfigurable PPAs and introduces the Superset Adder, a PPA with the forward tree of a Kogge-Stone combined with the reverse tree of a Brent-Kung. The advent of the Superset Adder is particularly useful as it subsumes all previously proposed base-2 PPAs with a fanout of two.

2. THE SUPERSET ADDER

2.1 Topology

Figure 1 shows the tree diagram of a Superset Adder. The top row, as previously discussed, is the pre-computation stage with each square creating their respective *generate* and *propagate* signals. The next three rows, also called the *forward tree*, contain the prefix nodes; each circle performs the aforementioned *dot operation*. The next two rows, also called the *reverse tree*, also contain prefix nodes. The last row is the post-computation stage, or summation stage, with each triangle creating their respective sum output.

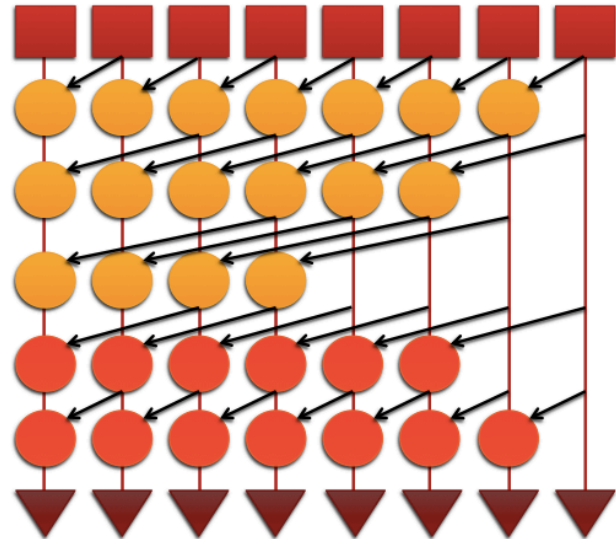


Figure 1. An 8-bit Superset Adder.

The key to reconfigurability can be found in the prefix nodes. Each node contains two multiplexers, or MUXes, that allow the output of the node to be one of two pairs of signals. Putting a '0' on the select line to the MUXes allows the node to pass its incoming *propagate* and *generate* signals, untouched. Conversely, setting the select line to '1' causes the MUXes to output the dot operator's computation. Being able to enable and disable prefix nodes allows the Superset Adder to achieve different configurations, bypass faults, increase performance, and decrease power. Determining the combination of nodes that give functioning adders can be difficult, but the Bailey's taxonomy reduces that complexity by organizing *regular* adders into the design space, where regularity implies periodicity. The following seven diagrams demonstrate the Superset Adder's ability to reconfigure into the seven *regular* parallel prefix adders.

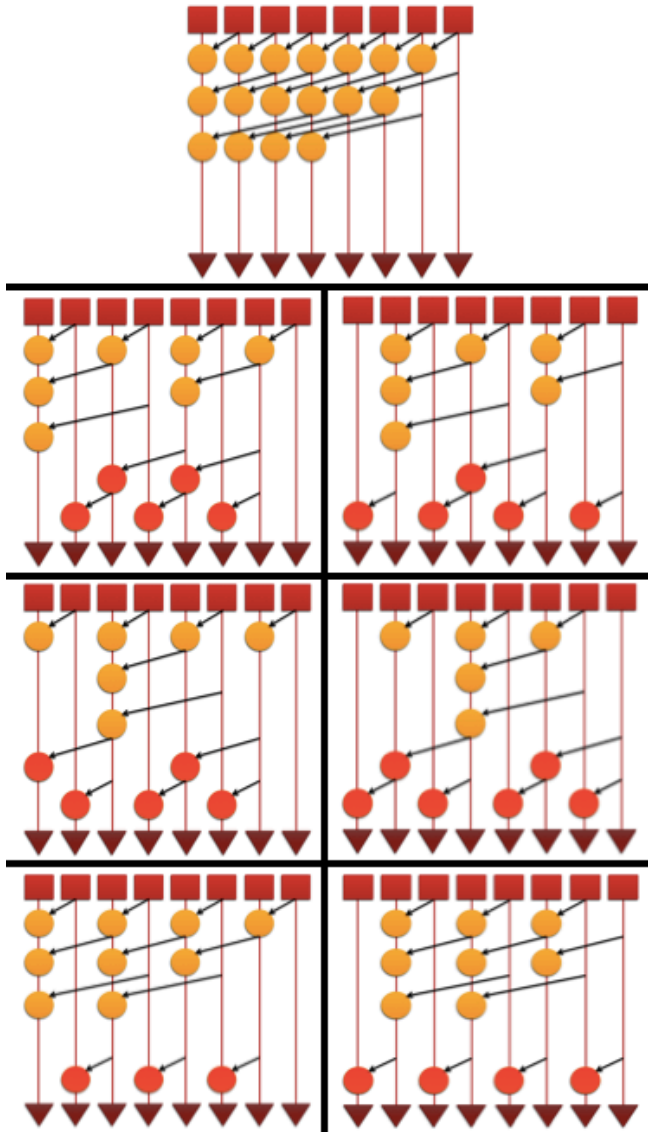


Figure 2. Tree graphs of the seven *regular* configurations

The Superset Adder configuration in the first row is a typical 8-bit Kogge-Stone – this would be useful in scenarios in which speed is of importance. The configurations in the second and third rows are variations of an 8-bit Brent-Kung – these would be useful in low-power scenarios. The last row of configurations demonstrates two variations of an 8-bit Han-Carlson – this is a hybrid Kogge-Stone/Brent-Kung that strikes a balance between delay and power. Numerous irregular configurations do and can exist, which may prove advantageous or even necessary in fault-isolation and error correction, but are further described in [6].

2.2 Logisim

For design verification and to make initial design in Cadence easier, we first created an 8-bit Superset Adder in Logisim, as seen in Figure 3.

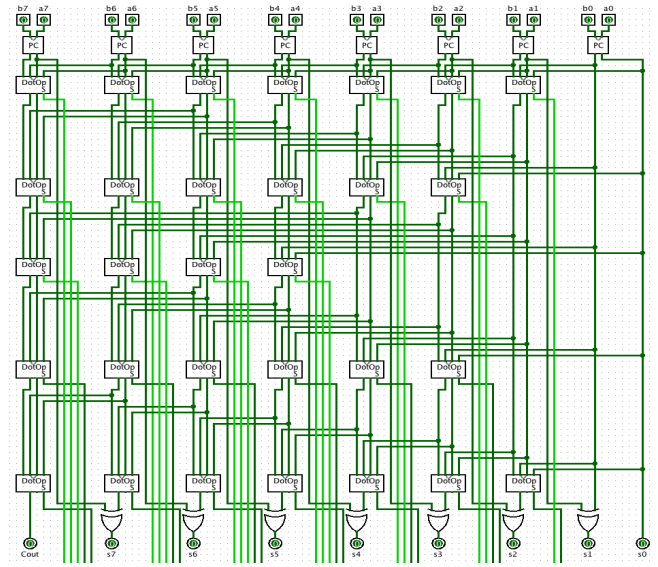


Figure 3. An 8-bit Superset Adder in Logisim

Again, the top row is the pre-computation row, the next three rows is the *forward tree* and consists of prefix nodes, the next two rows is the *reverse tree* and also consists of prefix nodes, and the last row is the summation stage, which consists of an XOR gate. Detailed gate level views can be seen in Figure 4.

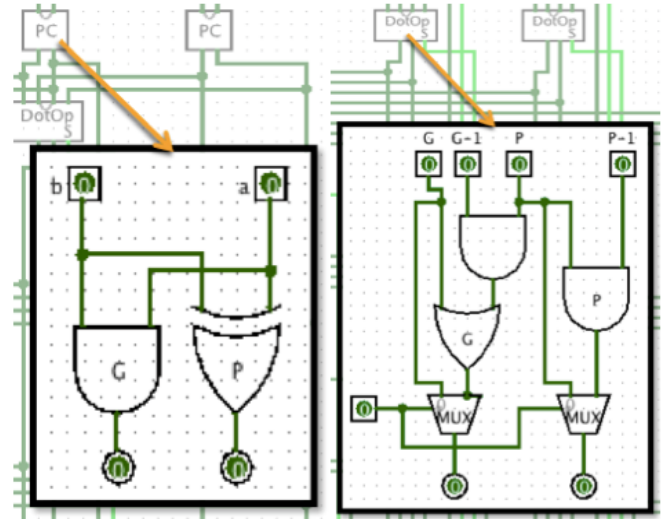


Figure 4. Pre-computation logic (left); MUXed prefix (right)

Additionally, preprogrammed configurations by use of an 8x1 MUX with 30-bit inputs were created in another sub-circuit topology. By feeding the 30-bit output of the MUX to the correct 30 select lines, this allowed the Superset Adder to easily switch between the Kogge-Stone, the four Brent-Kung variations, and the two Han-Carlson variations. This sub-circuit topology can be seen in Figure 5. After validating the seven configurations with various addition scenarios, we deemed it ready for realization in Cadence.

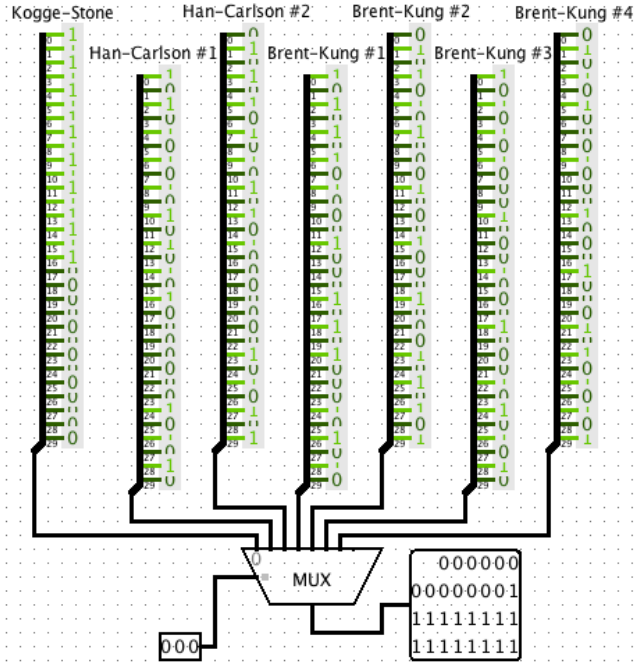


Figure 5. An 8-bit MUX with 30-bit inputs for reconfiguration

2.3 Cadence

Creating the Superset Adder in Cadence involved a similar process. Three main sub-circuits were created: the pre-computation block, the prefix node, and the summation block. We did, however, extend our design to 16-bits. The resulting design can be seen in Figure 6, which has been recolored for clarity.

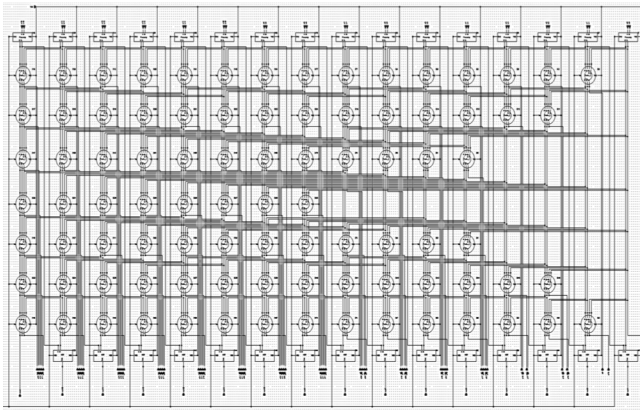


Figure 6. A 16-bit Superset Adder in Cadence

3. IMPROVING DELAY

3.1 Introduction

One common practice in speeding up ripple-carry adders is utilizing the inversion property. By inverting the inputs and carry-in on every other full adder, the delay on the critical path is reduced. We had also seen this done to Kogge-Stone adders and assumed, by extension, that it was possible in Brent-Kung and Han-Carlson adders. With the desire to reduce delay in the Superset Adder, we modified it to exploit this inversion property.

3.2 Logisim

To utilize inversion, many changes had to be made to the Superset Adder. The first change was in the pre-computation stage. Instead of an AND gate and XOR gate, we used a NAND gate and XNOR gate. This can be seen in Figure 7.

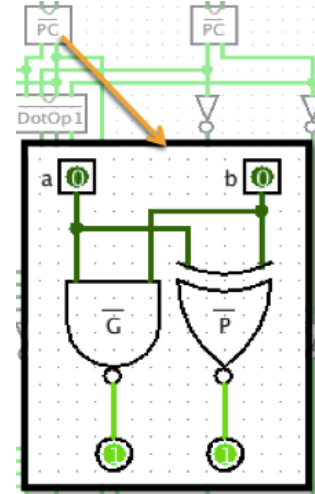


Figure 7. Pre-computation logic with inversion

The next change involved the prefix nodes. Two differing sets were created and used in an alternate fashion between rows; that is, the first row of prefix nodes were of one type and the following row of prefix nodes were of the alternate type, repeat. Both sets, as expected, have inverted outputs through the use of NAND and NOR gates. Inverters were also necessary on the *generate* and *propagate* signals when bypassing the dot operation to keep the topology in sync. These can be seen in Figure 8.

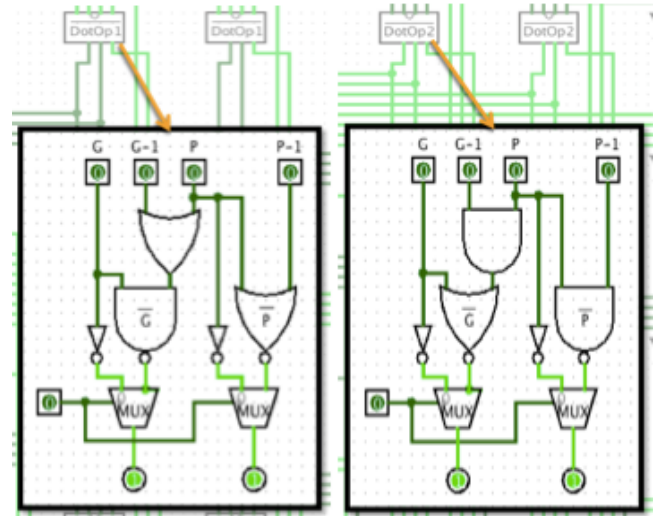


Figure 8. MUXed prefixed nodes with inversion

The finished Superset Adder “with inversion” can be seen in Figure 9. Notice, inverters were also necessary in areas where prefix nodes are not found, as to keep the topology, and specifically carries, in sync and valid.

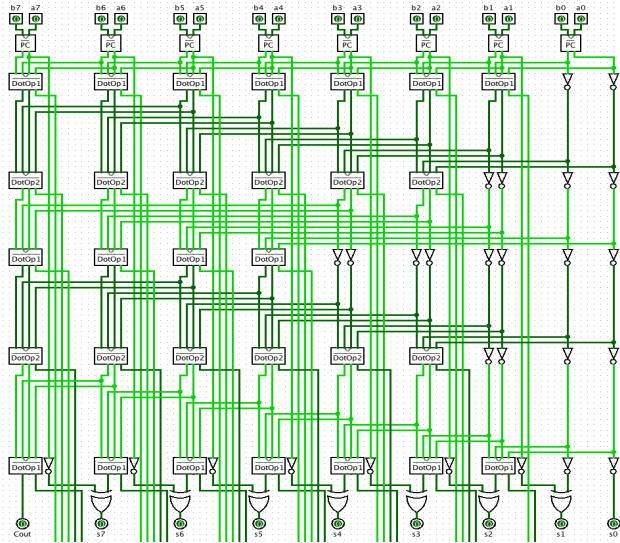


Figure 9. 8-bit Superset Adder “with inversion” in Logisim

3.3 Cadence

Once more, creating the Superset Adder “with inversion” in Cadence involved a similar process. Four main subcircuits were created: the pre-computation block, prefix node #1, prefix node #2, and the summation block. Just like the regular Superset Adder back in Figure 6, we extended the design to 16-bits. The resulting design can be seen in Figure 10, which has been recolored for clarity. A close-up of the alternating prefix nodes can be seen in Figure 11.

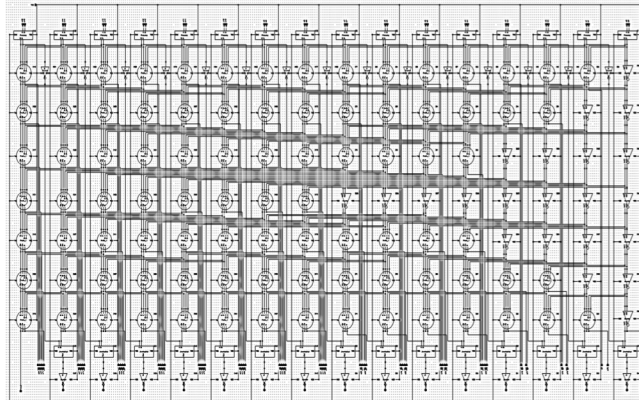


Figure 10. 16-bit Superset Adder “with inversion” in Cadence

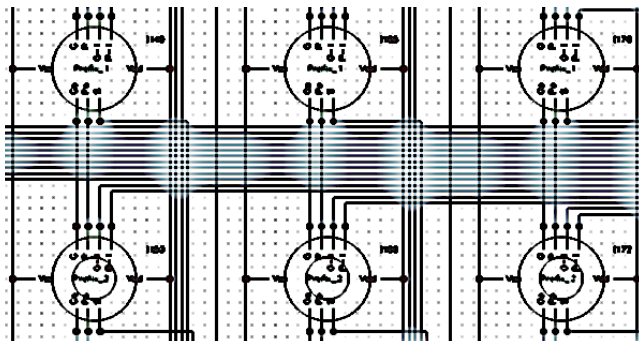


Figure 11. View of alternating prefix nodes in Cadence

4. REDUCING POWER

An industry-standard practice in reducing power consumption is power gating. The idea is to use a PMOS transistor as a header switch to shut off power to parts of the design in sleep or standby mode. We found this applicable to the Superset Adder because when a prefix node is being bypassed, it is simply a waste of power for it to perform its dot operation calculation concurrently. The solution was to power gate it. By splitting the VDD signal into the prefix node into two wires, we were able to send one wire to the multiplexers, as their operation is always a necessity, and the second wire to a PMOS transistor. A wire then connected the PMOS transistor to the VDD input of the gates. This can be seen in Figure 12.

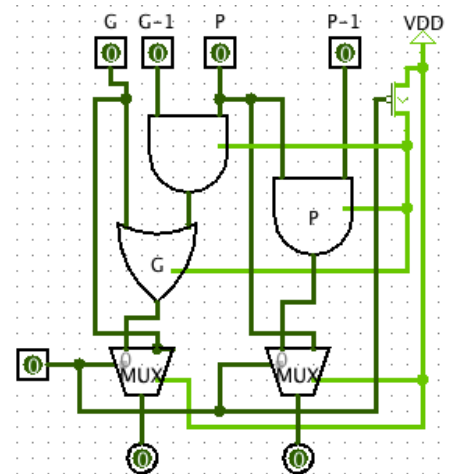


Figure 12. A power gated prefix node in Logisim

One change we had to make was to flip the MUXes so that the incoming *generate* and *propagate* signals went to the ‘1’ input of the MUX and the dot operator’s computation went to the ‘0’ input. This was necessary so that when the select line was ‘0’ to activate the prefix node, it was also turn on the PMOS switch, allowing VDD to power the logic gates.

5. RESULTS

Our first set of results, Table 1, demonstrates the trade-off in size when accounting for reconfigurability and fault-tolerance. The Superset Adder was found to be four times as large as an equivalent Kogge-Stone. This increase in area can be attributed to the reverse tree and numerous multiplexers. Naturally, utilizing inversion and power gating only increased area, as additional inverters and transistors were necessary.

Table 1. A comparison of areas in micrometers

	Area (μm)
Kogge-Stone	227.9
Superset	957.9
Superset (power gated)	966.0
Superset (inversion)	1036
Superset (inversion, power gated)	1044

For our next set of results, we tested the Superset Adder in a Kogge-Stone configuration by setting one input to 0000 0000 0000 0001 and the other input to 1111 1111 1111 1111; this creates a carry that exercises the critical path. Ignoring our base Kogge-Stone, we found the power gated Superset Adder to be the least power consuming and the Superset Adder “with inversion” to be the fastest.

Table 2. A Kogge-Stone comparison of delay and power

	Delay (ps)	Power (μ W)
Kogge-Stone	130.1	77.12
Superset	343.2	498.1
Superset (power gated)	374.4	368.5
Superset (inversion)	291.8	542.8
Superset (inversion, power gated)	308.1	458.5

Table 3 demonstrates a Brent-Kung configuration with the same input stimulation. We found the power gated Superset Adder “with inversion” to be marginally faster than the power gated Superset Adder (non-inversion), while the Superset Adder “with inversion” had the least delay.

Table 3. A Brent-Kung comparison of delay and power

	Delay (ps)	Power (μ W)
Superset	389.7	309.6
Superset (power gated)	429.4	282.6
Superset (inversion)	302.7	327.5
Superset (inversion, power gated)	353.3	282.2

Our last set of results is a Han-Carlson configuration with the same input setup. Our results show that the power gated Superset Adder had the least power consumption, while the Superset Adder “with inversion” was the fastest.

Table 4. A Han-Carlson comparison of delay and power

	Delay (ps)	Power (μ W)
Superset	345.8	354.1
Superset (power gated)	372.1	282.6
Superset (inversion)	295.2	379.4
Superset (inversion, power gated)	328.3	328.0

6. CONCLUSION

The Superset Adder is a novel solution for reconfigurability and fault-tolerance. What we have presented in this paper are ways to improve upon it. The utilization of the inversion property shows a drastic decrease in delay in any given configuration. The use of power gating also provides a large reduction in power consumption. In the case of the Kogge-Stone configuration, the Superset Adder “with inversion” and power gating was 10% faster and 8% more power efficient than its standard Superset Adder counterpart. In the case of the Brent-Kung configuration, it was 9.5% faster and 8.9% less power consuming. When considering the Han-Carlson configuration, it was 5.1% faster and 7.4% more power efficient. Thusly, we propose, that by using inversion and power gating simultaneously, the Superset Adder moves forward in its goal for high performance, low power applications where robustness is a necessity.

7. EXTENSIONS

The Superset Adder, however, may also benefit from additional modifications. The sizing of the header switch, the PMOS transistor, undoubtedly affects delay. We found a 0.02ns increase in speed in the Han-Carlson configuration when doubling its size from 90nm to 180nm. Using less area-intensive MUX designs, such as those with pass gates or transmission gates, can reduce the costly area of implementing multiplexers at each node. This, in turn, could also decrease delay and power consumption.

Moreover, our experiment used the same logic layout for each prefix node; some nodes, however, could be further reduced in area and power by removing the *propagate* logic (an entire AND gate) because the logic is not inherently necessary. We chose to keep the logic due to ease of implementation and future expandability (to 32-bits for example). Additionally, ideas such as increasing the radix and/or fanout, exploring the use of Null Convection Logic to eliminate glitches, and reducing the number of multiplexers and/or control signals when extending to other arithmetic datapaths have yet to be explored.

8. REFERENCES

- [1] Nassif, S., Mehta, N., Cao, Y. (2010). *A resilience roadmap. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1011–1016.
- [2] Rollins, N. et al. (2003). Evaluating TMR techniques in the presence of single event upsets. *Military and Aerospace Programmable Logic Devices (MAPLD)*, 63–67.
- [3] Harris, D. (2003). A taxonomy of parallel prefix networks. *Signals, Systems and Computers, 2003. Conference Record of the Thirty-Seventh Asilomar Conference*, 2, 2213–2217.
- [4] Ziegler, M., Stan, M. (2004). A Unified Design Space for Regular Parallel Prefix Adders. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1386–1387.
- [5] Guevara, M., Gregg, C. (2008). *Fault-tolerant, Real-time Reconfigurable Prefix Adder*. UVA, Charlottesville, Virginia.
- [6] Bailey, S., Stan, M. (2012). Proceedings from ISCAS, '12: *A new taxonomy for reconfigurable prefix adders*. Seoul, Korea.